Stochastic Modeling Formalisms for Dependability, Performance and Performability

Katerina Goševa – Popstojanova and Kishor Trivedi

Center for Advanced Computing and Communication
Department of Electrical and Computer Engineering
Duke University, Durham, NC 27708 – 0291, USA
{katerina, kst}@ee.duke.edu

Abstract. In this chapter, we discuss practical issues regarding analytical modeling of complex computer systems. We compare different model types in terms of their strengths and weaknesses for model construction, efficiency and accuracy of solution algorithms and the desired performance measures. The basic concepts of performance, dependability and performability modeling are introduced using the example of a multiprocessor system. With respect to combined modeling of performance and dependability two major problems are identified, largeness and stiffness, and a variety of approaches to deal with them are presented. Finally, techniques for high level model specification are briefly reviewed.

1 Introduction

Rapid advances in technology resulted in the proliferation of complex computer and communication systems that are used in different applications ranging from spacecraft flight-control to information and financial services. Dependability, performance, and performability evaluation techniques provide a useful method for examining the behavior of a computer or communication system right from the design stage to implementation and final deployment. The relative importance of performance and dependability requirements will differ depending on the system requirements and typical usage. Sometimes performance and dependability issues can be addresses separately, but sometimes their interactions and corresponding tradeoffs demand a measure that combines aspects of both.

Suppose that a multiprocessor system has to be designed. Some of the questions that need to be answered are the following. How much better will performance get by adding a processor? How would adding a processor affect the reliability of the system? Would this make system go down more often? If so, would an increase in performance outweigh the decrease in reliability?

The system designer has several options for predicting values: make an educated guess based on experience with previous similar systems; build prototypes and take measurements; use discrete event simulation to model the system; and construct analytic models of the system.

The actual measurement is the most direct method for assessing an existing system or a prototype, but it is not a feasible option during system design and

implementation phases. It is also sometimes impossible to assure by measurement that a system meets the design criteria. For example, in the case of highly reliable systems waiting for the system to fail enough times to obtain statistically significant sample would take years.

Discrete-event simulation (DES) is commonly used modeling technique in practice. It can capture system characteristics to the desired degree, and many software packages are available that facilitate the construction and execution of DES models. However, DES tends to be relatively expensive since it takes quite long time to run such models, particularly when results with high accuracy are required. Also, it is a non-trivial problem to simulate with high confidence scenarios entailing relatively rare events while others are occurring much more often.

Analytical modeling has proven to be an attractive cost-effective alternative in these cases. A model is an abstraction of a system that includes sufficient detail to facilitate an understanding of system behavior. To be useful, the model of current day complex computer and communication systems should reflect important system characteristics such as fault-tolerance, automatic reconfiguration and repair, contention for resources, concurrency and synchronization, deadlines imposed on tasks, and graceful degradation. Due the recent development in model generation and solution techniques, and the availability of software tools, large and realistic models can be developed and studied. A system designer has a wide range of different types of analytical models to choose from. Each type has its strengths and weaknesses in terms of accessibility, easy of construction, efficiency and accuracy of solution algorithms, and availability of software tools. The most appropriate type of model depends upon the complexity of the system, the questions to be studied, the accuracy required, and the resources available for the study.

Analytical models can be broadly classified into non-state space models and state space models. Reliability block diagrams, fault trees and reliability graphs are non-state space models commonly used to study dependability of systems. They are concise, easy to understand, and have efficient solution methods. However, realistic features such as non-independent behavior of components, imperfect coverage, non-zero reconfiguration delays, and combination with performance can not be captured by these models.

In the performance modeling, the examples of non-state space models are directed acyclic task precedence graphs and product form queueing networks. Directed acyclic task precedence graphs can be used to model concurrency for the case of unlimited resources. On the other hand, contention for resources can be represented by a class of queueing networks known as product form queueing networks (PFQN) for which efficient solution methods to derive steady state performance measures exist. However, they cannot model concurrency, synchronization, or server failures, since these violate the product form assumptions.

State space models enable us to overcome the limitations of the non-state space models in modeling complicated interactions between components and tradeoffs between different measures of interest. Although in this chapter we

concentrate on state space models, whenever it is possible we consider an alternative non-state space model.

Most commonly used state space models are Markov chains. They provide great flexibility for modeling dependability, performance, and combined dependability and performance measures. But the size of their state space grows much faster then the number of system components, making model specification and analysis difficult and error-prone process. One way to deal with large models is largeness-tolerance. A number of concise descriptions have evolved, and software tools that automatically generate the underlying Markov chain and provide effective methods for solution are now available. Many such high level specification techniques, queueing networks and stochastic Petri nets being the most prominent representatives, have been suggested in literature. Another way to deal with large models is to use techniques that avoid largeness, such as state truncation, state lamping, and model composition.

This chapter is organized as follows. First, we define Markov chains, and then introduce Markov reward models. Next, we demonstrate how a number of different pure dependability and pure performability measures can be derived by choosing the appropriate reward structure. We illustrate the combined performance and dependability analysis, and then examine two major difficulties that are encountered in the use of monolithic Markov models, namely, largeness and stiffness. Both problems of largeness and stiffness can be avoided by composing the overall model from a set of smaller non-stiff submodels. The overall solution is obtained by composing submodels solutions. Reward based performability analysis is an example of model composition approach; the performance submodel is solved and its results are passed as reward rates to the dependability submodel. Although in the chapter we use Markov reward models, the general concept of reward based modeling is not limited to a specific model type. Thus, we show how the Markovian constraints can be relaxed, and other paradigms such as semi Markov reward models or Markov regenerative reward models can be used as well. Finally, techniques for high level specification of the underlying computational model type are briefly reviewed.

Through the chapter we demonstrate the use of different model types and the derivation of a number of measures that may be of interest on the example of a multiprocessor system with n processors with a limited number of buffers m, in the presence of failure, reconfiguration, and repair.

2 Markov Reward Models: Definition and Measures

In this section we present a brief introduction to the concepts and notation of Markov chains and Markov reward models. Let $\{X(t), t \geq 0\}$ be a homogeneous finite state continuous time Markov chain (CTMC) with state space S and infinitesimal generator matrix $\boldsymbol{Q} = [q_{ij}]$. Let $P_i(t) = P\{X(t) = i\}$ denote the unconditional probability of the CTMC being in state i at time t, and the row vector $\boldsymbol{P}(t) = [P_1(t), P_2(t), \ldots, P_n(t)]$ represent the transient state probability vector of the CTMC. The transient behavior of the CTMC can be described by

the Kolmogorov differential equation:

$$\frac{d\boldsymbol{P}(t)}{dt} = \boldsymbol{P}(t)\,\boldsymbol{Q}, \qquad \text{given } \boldsymbol{P}(0) \tag{1}$$

where $\boldsymbol{P}(0)$ represents the initial probability vector (at time $t = 0$). The steady-state probability vector $\boldsymbol{\pi} = \lim_{t\to\infty} \boldsymbol{P}(t)$ satisfies:

$$\boldsymbol{\pi}\boldsymbol{Q} = 0, \qquad \sum_{i\in S} \pi_i = 1. \tag{2}$$

In addition to transient state probabilities, sometimes cumulative probabilities are of interest. Define $\boldsymbol{L}(t) = \int_0^t \boldsymbol{P}(u)\,du$; then $L_i(t)$ denotes the expected total time the CTMC spends in state i during the interval $[0, t)$. $\boldsymbol{L}(t)$ satisfies the differential equation:

$$\frac{d\boldsymbol{L}(t)}{dt} = \boldsymbol{L}(t)\,\boldsymbol{Q} + \boldsymbol{P}(0), \qquad \boldsymbol{L}(0) = 0. \tag{3}$$

With these definitions, most of the interesting measures can be defined. CTMC with absorbing states deserves additional attention. Here, the measures of interest are based on the time a CTMC spends in non-absorbing states before an absorbing state is ultimately reached. For that purpose the state space $S = A \cup T$ is partitioned into the set A of absorbing states and the set T of non-absorbing (transient) states. Let $\boldsymbol{Q}_T$ be the submatrix of $\boldsymbol{Q}$ corresponding to the transitions between transient states. Then the time spent in transient states before absorption can be calculated by $\boldsymbol{L}_T(\infty) = \lim_{t\to\infty} \boldsymbol{L}_T(t)$ restricted to the states of the set T. The mean time to absorption (MTTA) can be written as $MTTA = \sum_{i\in T} L_i(\infty)$.

Assigning rewards to states or to transitions between states of CTMC defines Markov reward model (MRM). In the former case rewards are referred to as reward rates and in the letter as impulse rewards. In this chapter we consider state-based rewards only. Let the reward rate r_i be assigned to state i. Then, the random variable $Z(t) = r_{X(t)}$ refers to the instantaneous reward rate of the MRM at time t. The accumulated reward over the interval $[0, t)$ is given by

$$Y(t) = \int_0^t Z(u)\,du = \int_0^t r_{X(u)}\,du. \tag{4}$$

Based on the definitions of $X(t)$, $Z(t)$, and $Y(t)$, which are non-independent random variables, various measures can be defined. The most general is the distribution of the accumulated reward over time $[0, t)$, that is, $P\{Y(t) \le y\}$ which is difficult to compute for unrestricted models and reward structures [5]. The problem is considerably simplified if we restrict to the expectations and other moments of random variables. Thus, the expected instantaneous reward rate can be computed from

$$E[Z(t)] = \sum_{i\in S} r_i\,P_i(t) \tag{5}$$

and the expected reward rate in steady-state (when the underlying CTMC is ergodic)

$$E[Z] = \sum_{i \in S} r_i \, \pi_i. \tag{6}$$

To compute the expected accumulated reward we use

$$E[Y(t)] = \sum_{i \in S} r_i \, L_i(t). \tag{7}$$

For models with absorbing states, the limit as $t \to \infty$ of the expected accumulated reward is called the expected accumulated reward until absorption

$$E[Y(\infty)] = \sum_{i \in T} r_i \, L_i(\infty). \tag{8}$$

Given the MRM framework, the next question that arises is "What are the appropriate reward rate assignments?". The reward structure clearly depends on whether we are interested in dependability, performance or composite dependability and performance measures. In the next section we will illustrate the use of this general framework for deriving a number of different measures.

3 Separate Analysis of Dependability and Performance

We begin by introducing a dependability model of multiprocessor system that considers failure/repair behavior and derive a number of dependability related measures by choosing appropriate reward structures. Next, the performance model of the multiprocessor system that describes the arrival and service completion of the jobs is presented and we demonstrate that the concept can also be used to derive performance measures of interest.

3.1 Dependability Model

Reliability, availability, safety and related measures are collectively known as dependability. Thus, dependability modeling encompasses failure, reconfiguration, and repair related aspects of system behavior. We present dependability model of a multiprocessor system with two processors, adapted from [5]. Each processor is subject to failures so that its MTTF is $1/\gamma$. A processor failure is covered with probability c, that is, not covered with probability $1 - c$. A covered failure is followed by a brief reconfiguration period, the average reconfiguration time being $1/\delta$. An uncovered failure is followed by a reboot, which requires a longer time to take place; the average reboot time being $1/\beta$, $(1/\beta > 1/\delta)$. In either case the failed processor needs to be repaired, with mean time to repair being $1/\tau$. The other processor continues to run and provides service normally, that is, the system comes up in a degraded mode. Should the other processor fail before the first one is repaired, the system becomes out of service until the repair of the one of the processors is completed. Only one processor can be repaired at

a time. Neither reboot nor reconfiguration is performed when the last processor fails. It is assumed that no other event can take place during a reconfiguration or reboot. The justification for this assumption lies in the fact that in practice the reconfiguration and reboot times are extremely small compared to the time between failures and repair times. If all the times are assumed to be independent exponentially distributed random variables, then the multiprocessor system can be modeled by the CTMC shown in Fig.1.

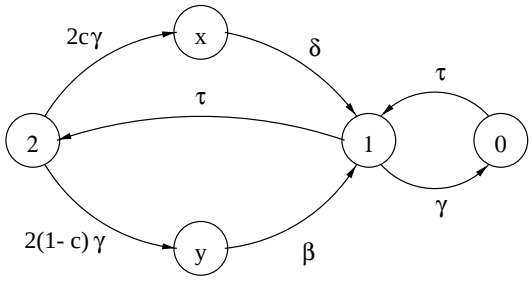

Fig. 1. Dependability model of multiprocessor system

System Availability Measures. System availability measures show the likelihood that the system is delivering adequate service, or equivalently, the proportion of potential service actually delivered. These measures fit best with the system where brief interruptions in system operation can be tolerated, but no significant annual outage. For example, commercial telephone switching systems and database systems are designed to provide high system availability over a long periods of time.

The most simplest availability measures are based on a binary reward structure. Assuming for the model in Fig.1 that one processor is sufficient for the system to be up, the state space S can be partitioned into a set of up states $U = \{2, 1\}$ and set of down states $D = \{x, y, 0\}$, that is, $S = U \cup D$. A reward rate 1 is attached to the up states and a reward rate 0 to down states. It follows that the probability that the system is up at a specific time t, that is, instantaneous availability is given by

$$A(t) = E[Z(t)] = \sum_{i \in S} r_i P_i(t) = \sum_{i \in U} P_i(t) = P_1(t) + P_2(t). \tag{9}$$

Steady-state availability, on the other hand, is given by

$$A = E[Z] = \sum_{i \in S} r_i \pi_i = \sum_{i \in U} \pi_i = \pi_1 + \pi_2. \tag{10}$$

The interval availability provides a time average value

$$\bar{A}(t) = \frac{1}{t}E[Y(t)] = \frac{1}{t}\sum_{i\in S} r_i L_i(t) = \frac{1}{t}\sum_{i\in U} L_i(t) = \frac{1}{t}[L_1(t) + L_2(t)], \qquad (11)$$

that is, the expected fraction of time from the system start until time t that the system is up. Note that unavailability can be calculated with a reverse reward assignment to that for availability.

Instantaneous, interval, and steady-state availabilities are the fundamental measures to be applied in this context, but there are other availability related measures that do not rely on the binary reward structure [5].

System Reliability Measures. System reliability measures emphasize the occurrence of undesirable events in the system. These measures are useful for systems where no down time can be tolerated, such as flight control systems. System reliability represents the probability of uninterrupted service exceeding a certain length of time, that is, $R(t) = P\{T > t\}$. What kind of event is to be considered as an interruption depends on the application requirements. In computing reliability for our example, we consider three different variants. In the most restrictive application context (variant 1) any processor failure is considered a system failure, that is, both reconfiguration and reboot are considered as interruption. In variant 2 uncovered processor failure and a failure of the last remaining functional processor is considered to be a system failure, that is, reconfiguration may be tolerated. As a variant 3 we assume that both reconfiguration and reboot can be tolerated, that is, only the failure of the last remaining functional processor is considered as a system failure. The three possible variants are captured in Fig.2. Note that for the purpose of reliability modeling the CTMC in Fig. 1 has been adopted by making all system down states absorbing that reflect the fact that they are considered as representing interruptions. Again, a binary reward structure is defined that assigns reward rates 1 to up states and reward rates 0 to (absorbing) down states. It follows that reliability can be computed by

$$R(t) = E[Z(t)] = \sum_{i\in S} r_i P_i(t) = \sum_{i\in U_j} P_i(t) \qquad (12)$$

where U_j represents the set of corresponding up states of the variant j. The three different reliability functions give the probabilities that in time interval $[0, t)$ there is:

- no outage $R_1(t) = P_2(t)$,
- no outage due to uncovered failure or lack of processors $R_2(t) = P_2(t) + P_x(t) + P_1(t)$,
- no outage due to lack of processors $R_3(t) = P_2(t) + P_x(t) + P_y(t) + P_1(t)$.

In the case of binary reward structure the system mean time to failure (MTTF) is just an MTTA for the CTMC with absorbing states, that is,

$$MTTF = MTTA = E[Y(\infty)] = \sum_{i\in U_j} L_i(\infty). \qquad (13)$$

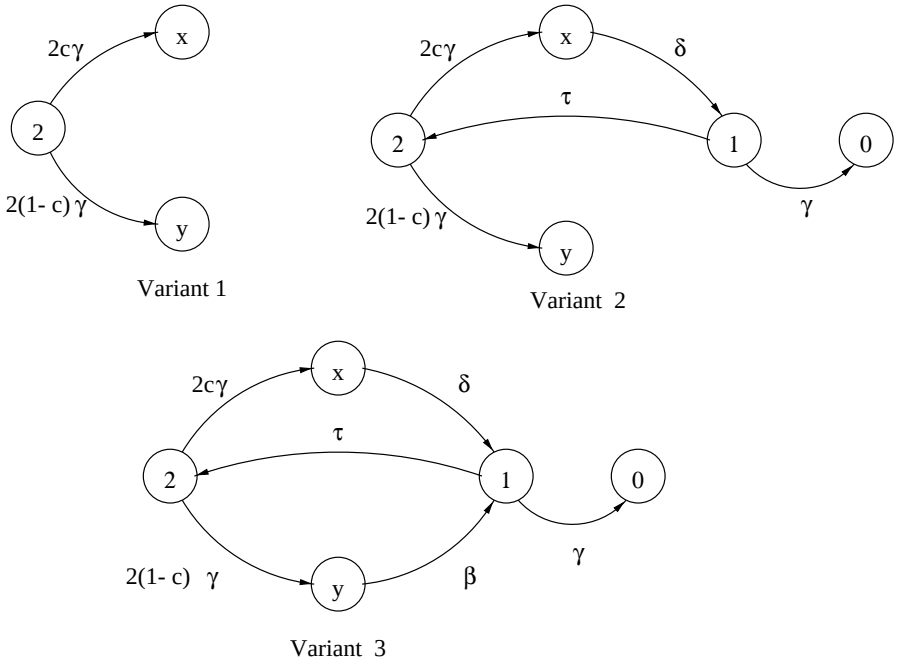

Fig. 2. Model variants with absorbing states capturing reliability requirements

3.2 Performance Model

The MRM framework can also be used in pure (failure-free) performance models to conveniently describe performance measures of interest. In many computer performance studies, expected throughput, mean response time, or utilization are the most important measures. These measures can easily be specified by means of appropriate reward functions. To illustrate the reward assignment for these measures we consider $M/M/1/m$ queue as a performance model of a single processor system. Imagine jobs (tasks, customers) are arriving at the system with exponentially distributed interarrival times with mean $1/\lambda$. In the system they compete for the service from a single server station. Since service is exclusively received by each job, if more then one job is in the system at the same time, the others have to wait in queue until their turn comes. Service times are independent exponentially distributed with mean $1/\mu$. To keep the example simple, we limit the maximum number of customers in system to three. The system is described by the CTMC shown in Fig. 3. Every state in $S = \{0, 1, 2, 3\}$ represents the number of customers in the system. A state transition occurs if a new job arrives or if a job being served completes the service.

The throughput characterization can be achieved by assigning the state transition rate corresponding to departure from a queue (service completion) as a reward rate to the state where the transition originates. It follows that the reward assignment for our example is $r_i = \mu$ for $i = 1, 2, 3$ and $r_0 = 0$. With this

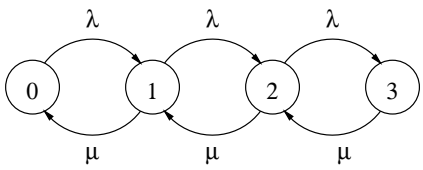

Fig. 3. Performance model (CTMC of M/M/1/3 queue)

reward structure we can compute the steady-state throughput

$$\lambda = E[Z] = \sum_{i \in S} r_i \, \pi_i = \mu[\pi_1 + \pi_2 + \pi_3]. \tag{14}$$

The mean number of jobs in the system can be computed by assigning to each state the reward rate equal to the number of jobs in the system in that state, that is, $r_i = i$. It follows that the mean number of jobs in steady-state is

$$\bar{K} = E[Z] = \sum_{i \in S} r_i \, \pi_i = \pi_1 + 2\pi_2 + 3\pi_3. \tag{15}$$

Mean response time measures of queueing system can be calculated from the mean number of jobs with the help of Little's theorem [12] as

$$\bar{T} = \frac{1}{\lambda} \bar{K}. \tag{16}$$

Finally, the utilization measures can be computed based on a binary reward assignment. Thus, if the particular resource is occupied in a given state, reward rate 1 is assigned, otherwise reward rate 0 indicates the idleness of the resources. With reward structure $r_i = 1$ for $i = 1, 2, 3$ and $r_0 = 0$ the utilization becomes

$$\rho = E[Z] = \sum_{i \in S} r_i \, \pi_i = \pi_1 + \pi_2 + \pi_3. \tag{17}$$

4 Composite Performance and Dependability Analysis

In the previous section, we saw that the goal of an availability model is to determine the fraction of time spend in up states and that the reliability measures answer the question "Assuming that the system is working initially, how long will it continue to work without interruptions?". We also saw that the performance model is developed when we are interested in the level of productivity of a system, or in answering the question "How well is the system working, given that it does not fail?".

Modeling any system with either a pure performance model or a pure dependability model can lead to incomplete or even misleading results. Analysis from pure performance viewpoint tends to be optimistic since it ignores the failure/repair behavior of the system. On the other hand, pure dependability

analysis is carried out in the presence of component failures, disregarding the different performance levels in different configurations, that is, tend to be too conservative. Complex systems are designed to continue working even in the presence of failures, guaranteeing a minimum level of performance. These systems are gracefully degradable and have redundant components that are all used at the same time to increase the system processing power. If a component in a degradable system fails, the system itself detects the failure and reconfigures, reaching a degraded state of operation in which it continues to provide service, but at a reduced capacity. A degradable system can have several degraded operational states between being fully operational and having completely failed. Each state provides different performance level. In such cases, pure performance or pure dependability models do not capture the entire system behavior. The measures of interest for a degradable system aim to answer the following question "What is the expected performance of the system at time t including the effects of failure, repair, contention for resources and so on ?".

Several different types of interactions and corresponding tradeoffs have prompted the researchers to develop methods for combined evaluation of performance and dependability. The first approach is to combine the performance and dependability behavior into an exact monolithic model. Let us consider our example of a single processor system. So far we have presented separate dependability and performance models for this example. Now we will generalize the $M/M/1/m$ queueing model presented in Fig.3 by allowing for the possibility that a server could fail, and that a failed server could be repaired. Again, let the job arrival rate be λ and the job service rate be μ. Let the processor failure rate be γ and the processor repair rate be τ. This system can be modeled using an irreducible CTMC shown in Fig.4 with the state space $S = \{(i,j), 0 \leq i \leq m, j = 0,1\}$, where i denotes the number of jobs in the system and j the number of functioning servers.

Two distinct problems arise from this monolithic approach: largeness and stiffness. The largeness problem can be tolerated to some extent by using high level specification techniques, such as generalized stochastic Petri nets (GSPN), and automated methods for generating the Markov chain. GSPN model that is equivalent to the Markov model in Fig.4, adopted from [14], is shown in Fig.5. The cycle in the upper part of the figure is a representation of an $M/M/1/m$ queue. The lower cycle models a server that can fail and be repaired. The inhibitor arc from place *server-down* to the transition *service* reflects the fact that jobs cannot be served while the server is not functioning. The number in each place is the initial number of tokens in the place. All of the transitions are timed, and each transition's rate is shown below the transition.

The GSPN description of the model is concise and allows us to vary the values of m without changing the models structure. However, since no model reduction is employed the underlying CTMC is very large. Therefore, large model tolerance must also apply to storage and solution of the model, that is, the appropriate data structures for sparse matrix storage and sparsity preserving solution methods must be used.

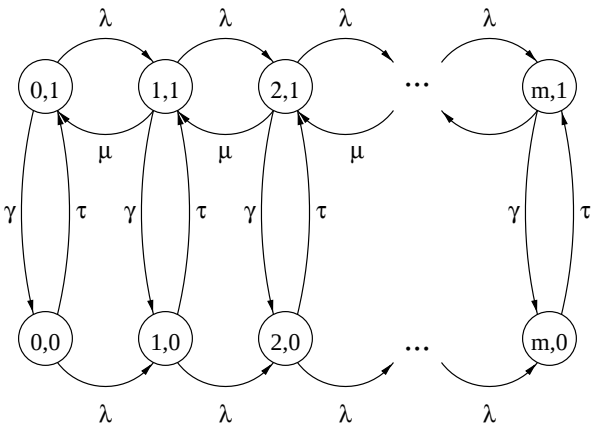

Fig. 4. CTMC model of a $M/M/1/m$ queueing system subject to server failure and repair (Monolithic model)

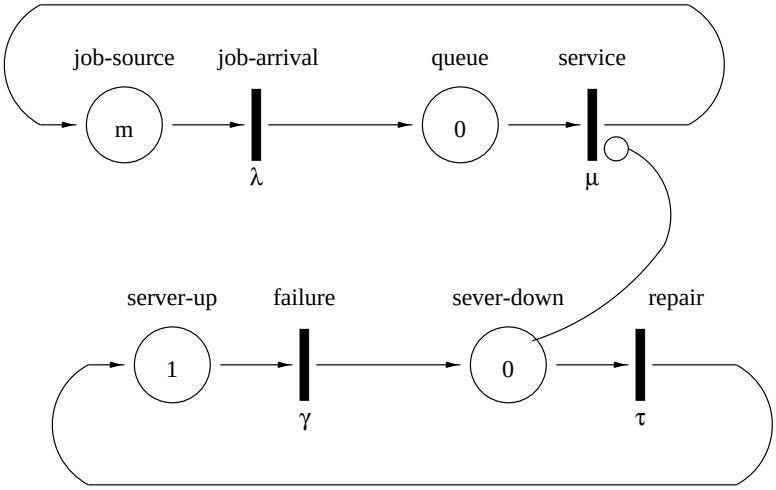

Fig. 5. GSPN model for queue with server failure and repair

Stiffness is another undesirable characteristic of monolithic models. It is due to the different orders of magnitude (sometimes 10^6 times) between the rates of occurrence of performance-related events and the rates of the rare, failure-related events. Stiffness leads to difficulty in the solution of the model and numerical instability. Current research in the transient solution of stiff Markov models follows two main lines: stiffness-tolerance and stiffness-avoidance. The first one is aimed at employing solution methods that remain stable for stiff models (see for extensive survey [5] and references therein). In the second approach, stiffness is eliminated from a model by solving a set of non-stiff submodels. One such technique based on aggregation proceeds by decomposing the original model into smaller submodels [3]. An approximate solution can be obtained by first solving the submodels in isolation (the aggregation step) and then combining the submodel solutions into the solution of the original model (the disaggregation step). A notable property of the decomposition technique is that besides reducing the size of the submodels on which transient analysis is carried out, it also eliminates the stiffness, making the application of standard numerical methods more efficient.

The aggregation technique applies when stiffness arises from the presence of rates belonging to two well separated sets of values in the transition rate matrix of the Markov chain. These rates are accordingly classified into fast and slow rates. In our example it is reasonably to assume that the rates of occurrence of performance-related events λ and μ differ by orders of magnitude relative to the rates of failure/repair-related events γ and τ. Usually, repair of a failed unit takes much longer than traffic-related events in computer systems. This condition is even more relevant for failure events that are relatively rare. Thus the transition rates λ and μ can be classified as being fast, and γ and τ as slow. States of the Markov chain are also classified into fast and slow states; a state is fast if at least one outgoing transaction has a fast rate, otherwise the state is slow. The state space is partitioned according to the classification scheme applied to the rates, into a set of slow states $S_0 = \{(m,0)\}$, a set of fast recurrent states $S_1 = \{(0,1),(1,1),\ldots,(m,1)\}$, and a set of fast transient states $S_2 = \{(0,0),(1,0),\ldots,(m-1,0)\}$. An appropriate aggregation algorithm is separately applied to each subset of fast states. The model of Fig.4 after aggregation of fast recurrent subset into macro-state 1 is shown in Fig.6a, while the final aggregated macro-state chain after elimination of the fast transient states is presented in Fig.6b. Thus, transient approximate solution is obtained by integrating a smaller, non-stiff set of linear differential equations. Next, the disaggregation must be performed to provide an approximation to the transient probability vector $P(t)$. Then, all measures of interest can be calculated from the transient state probabilities. For the detailed algorithm the reader is referred to [5]. The empirical results presented there support the assumption of the approximation being better for stiffer models.

Both the problems of largeness and stiffness can be avoided by using hierarchical model composition. Occurrence rates of failure/repair events are several orders of magnitude smaller than the job arrival/service rates. Consequently, we

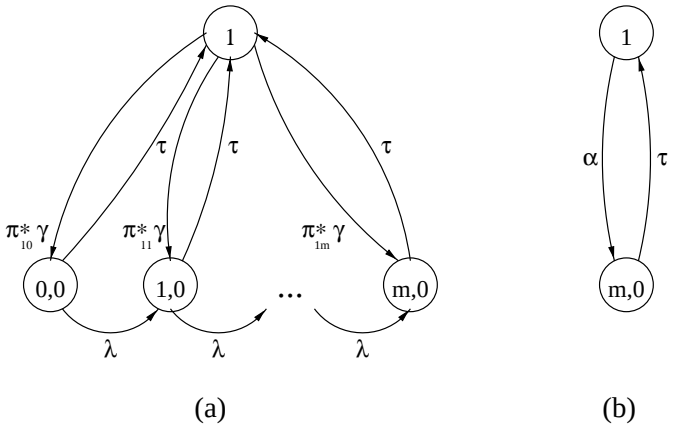

Fig. 6. (a) The model of Fig.4 after the aggregation of the fast recurrent subset into macro-state 1. (b) Final aggregated macro-state chain after elimination of the fast transient subset

can assume that the system attains a (quasi-) steady state with respect to performance related events between successive occurrences of failure/repair events, that is, the performance measures would reach a stationary condition between changes in the system structure. This leads to a natural hierarchy of models. The structure state model is the higher level dependability model representing the failure/repair processes. For each state in the dependability model, there is a reward model, which is a performance model for the system with a given, stationary structural state. Several authors have used the letter concept in developing techniques for combined performance and dependability analysis. Early and defining work in this field was done by Beaudry [2] who computed the computational availability until failure. Meyer [11] proposed a conceptual framework of performability, that enable us to characterize degradable systems in terms of their ability to provide a given amount of useful work in a given period of time. Most of the proposed approaches for performability modeling can be brought under the broad framework of Markov reward processes [8] summarized in Sect.2.

In the next section we present the case study (adapted from [13]) which illustrates the use of hierarchical model composition on the example of a multiprocessor system.

5 Case Study of a Multiprocessor System

For the case study we consider a multiprocessor system with n processors (subject to failure and repair) with a limiting number of buffers m. First, we determine the optimal number of processors based on either pure performance or pure dependability measures. Then, we consider the total loss probability which combines performance and availability measures as the most appropriate measure of

system effectiveness. The optimal configuration in terms of number of processors is shown to be a function of the chosen measure of system effectiveness.

5.1 Sizing Based on Performance

The performance model is an $M/M/n/m$ queue which can be modeled using a birth and death type Markov chain, as shown in Fig.7 for the case where $n = 5$ and $m = 100$.

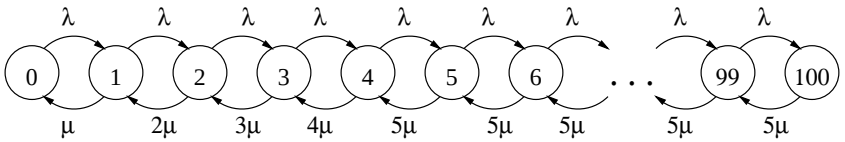

Fig. 7. Birth and death type Markov chain for the $M/M/n/m$ queueing system

The state space of the associated Markov chain grows fast as the size of a queueing network increases. As discussed in Sect.4, high level specification techniques, such as queuing networks or stochastic Petri nets can be used to describe the CTMC in Fig.7. Thus, it is possible to convert this open queueing model into a closed product-form queueing network, which is shown in Fig.8. This model contains two stations. Station *mp* is the processor station with n processors, each having service rate μ. The other station is *source*, which represents the job source with rate λ. Because there is a limited number of buffers available for queueing the jobs, the closed product-form network with a fixed number m of jobs is chosen.

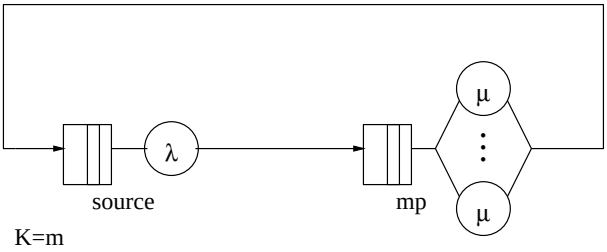

Fig. 8. Closed PFQN model of the $M/M/n/m$ queueing system

PFQN is a useful class of queueing networks that can be analyzed without generating the underlying state space for the whole network. In other words, they belong to the class of non-state space models. PFQN can be used as efficient method for the large model tolerance since many algorithms for exact and approximate solutions for steady-state performance measures exist. However,

PFQN cannot be used to model concurrency, synchronization, or server failures, since these violate the product form assumptions.

It is also possible to model the $M/M/n/m$ queue using the GSPN in Fig.9. The initial number *nproc* of tokens in place *proc* means that there are *nproc* processors available. When a new job arrives in place *buffer*, a token is removed from place *proc*. Jobs arrive at the system when transition *arr* fires. There is a limitation for new jobs entering the system caused by the inhibitor arc from place *buffer* to transition *arr*. Thus *arr* can only fire when the system is not already full. There can be only m jobs in the system altogether, *nproc* being served (in place *serving*) and m – *nproc* in place *buffer*. The firing rates are λ for transition *arr* and $k\mu$ for transition *service*. Here k is the number of tokens in place *serving* and the notation for this marking dependent firing rate in Fig.9 is $\mu\#$.

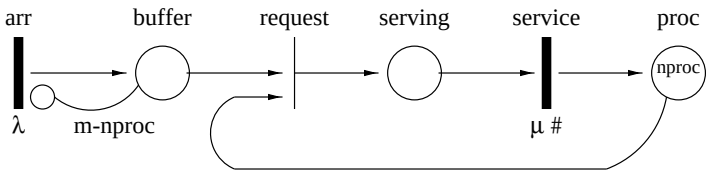

Fig. 9. GSPN model of the $M/M/n/m$ queueing system

The GSPN model of the $M/M/n/m$ queue is much more concise than the Markov model. The Markov chain model has as many states as there are potential jobs in the system m. If we use the Markov model to get results for different values of n and m, we have to build a new Markov model for each pair of values. The GSPN in Fig.9 will let us vary the value of n and m without changing the model structure. We just need to change the initial marking in place *proc* and the multiplicity of the inhibitor arc. It is important to note that the smaller size of the GSPN model does not mean that the model analysis is correspondingly easier. While increasing n and m does not change the size of GSPN model, it does make the underlying CTMC bigger as already discussed in Sect.4.

Returning to our example of sizing of multiprocessor system based on performance, as an appropriate performance measure we use the job loss probability due to a system being full or too slow. The closed form solutions are available to calculate the probability of a job being rejected because the buffers are full $q_m(n)$ [7]. For an accepted job, define the response time random variable to be $R_n(m)$. The closed form formula for the response time distribution can be derived based on the formula for waiting time given in [7]. If there is a deadline d imposed on the job response times then we can find the probability of system being too slow, that is, a late completion of a job due to deadline violations as $P\{R_n(m) > d\}$.

The job loss probability reflects the effect of job rejection due to buffer full as well a deadline violation of accepted job (system slow)

$$lp(n) = q_m(n) + [1 - q_m(n)]\, P\{R_n(m) > d\}. \tag{18}$$

The equations for $q_m(n)$ and $P\{R_n(m) > d\}$, and the numerical results for the loss probability could be found in [13]. Since the loss probability of a task is monotonically decreasing in the number of processors, the conclusion from the model in this subsection is that the performance of fault free system improves as we increase the number of processors in the multiprocessor system.

5.2 Sizing Based on Availability

For dependability analysis we consider a multiprocessor system with n processors subject to failure and repair. Reliability block diagram and fault tree model of this system are shown in Fig.10a and Fig.10b respectively. These models belong to the class of non-state space models specialized for dependability analysis. They are concise, easy to understand, and have efficient solution methods under the assumption that components failure and repair times are independently distributed, and there are enough repair resources to repair all components at the same time, if necessary. However, non-state space models do not allow us to model realistic features such as shared repair facilities, imperfect coverage, and non-zero reconfiguration delays.

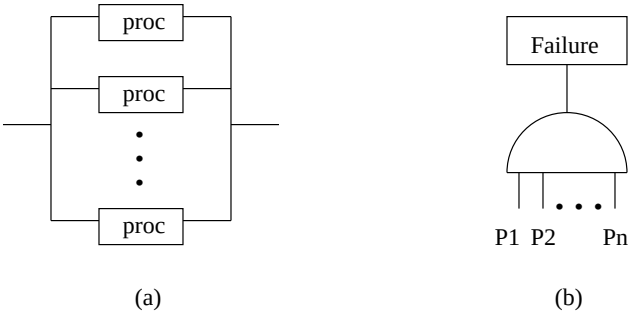

Fig. 10. Non-state space dependability models of a multiprocessor system (a) Reliability block diagram model (b) Fault tree model

By contrast, state space models enable us to account for such details easily, as we have shown in Sect.3. Thus, for the availability model we consider the CTMC presented in Fig.1 for the general case with n processors. The Markov chain for this system is shown in Fig.11. In state i, $1 \leq i \leq n$, the system is up with i processors functioning, and $n-i$ processors waiting for on-line repair. The processors share the repairing facility and only one processor can be repaired at a time. Following a covered failure the system is undergoing a reconfiguration in

states x_{n-i}, $i = 0, \ldots, n-2$, while an uncovered failure is followed by a reboot in states y_{n-i}, $i = 0, \ldots, n-2$. In state 0, the system is down waiting for off-line repair. If we assume that the system is down while a reconfiguration, reboot or an off-line repair is in progress, the steady-state availability $A(n)$ defined as a function of n is given by

$$A(n) = \sum_{i=0}^{n-1} \pi_{n-i}. \tag{19}$$

The equations for steady-state probabilities π_{n-i}, $\pi_{x_{n-i}}$ and $\pi_{y_{n-i}}$ could be found in [13]. Under the assumptions that the coverage is not perfect and there is non-zero reconfiguration delay, the unavailability $1 - A(n)$ is minimized at a small number (two) of processors [13].

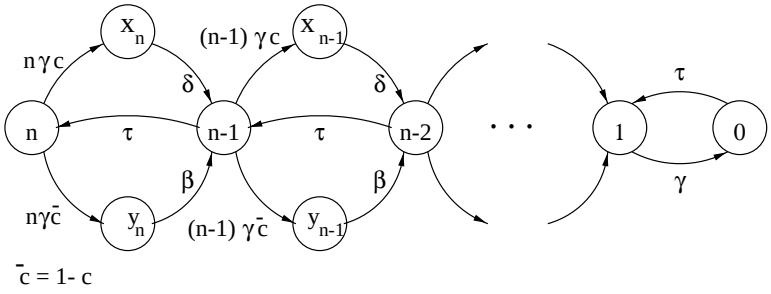

Fig. 11. CTMC for computing availability of a multiprocessor system

5.3 Sizing Based on Performability

The goal of an availability model is to determine the fraction of time spent in up states. If we use only the result of previous subsection, we could come to the conclusion that the best system configuration is one with two processors. However, increasing the number of processors improves system performance. It is clear that measures such as availability and reliability do not reflect the increased performance due to the increasing number of processors. The most appropriate measure of system effectiveness that reflects both fault-free behavior and behavior in the presence of failures is the total loss probability due to a system being full or too slow or system being down.

For the purpose of deriving the combined performance and availability measure we use hierarchical model composition, that is, a Markov reward model, which avoids both problems of largeness and stiffness. The higher level model (the structure state model), is the Markov chain in Fig.11 which represents the state of the system with regard to failures and repairs. The total loss probability is obtained by assigning a reward rate to each state, equal to the probability that a task is rejected in that state. In other words, the rewards are assigned to the structural states as follows:

- For all up states with i $(1 \leq i \leq n)$ processors functioning the lower level model which captures the performance of the system is an $M/M/i/m$ queue. Thus, for up states we set the reward rate $r_i = lp(i)$ which is the probability of job rejection due to the buffer full or to deadline violation of accepted job (system slow).
- For all down states x_{n-i}, y_{n-i} $(0 \leq i \leq n-2)$ and state 0 the reward rate assigned is 1, since an arriving job is always rejected when the system is unavailable.

It follows that the total loss probability is the expected steady-state reward rate given by

$$TLP(n) = E[Z] = \sum_{i=0}^{n-1} lp(n-i)\, \pi_{n-i} + \sum_{i=0}^{n-2} (\pi_{x_{n-i}} + \pi_{y_{n-i}}) + \pi_0. \qquad (20)$$

Note that the first term in (20) is the loss probability due to the system full or too slow. The last two terms give the loss probability due to the system being down which is equal to unavailability $1 - A(n)$.

Using the reward structure just discussed and (20), [13] compute the optimal number of processors. For example, it is shown that if $m = 10$ and $\lambda = 80$ per second the optimal number of processors is 3 for one second deadline on task response time, 4 for 0.1 second deadline on task response time, and greater then 8 for 0.01 second deadline on task response time. There is an obvious tradeoff with availability criteria which imposes optimal number of two processors.

The analysis of variation in the total loss probability as a function of number of processors n for different values of the task arrival rate λ ($d = 0.1$ second and $m = 10$) show that for very small values of $\lambda = 1/sec$, the TLP is essentially equal to system unavailability, hence the optimal number of processors is two. For larger values of λ, the rejection probability due to system being full and too slow starts to play a dominant role in increasing the optimal number of processors.

6 Relaxing the Markovian Constraints

A major objection to the use of Markov models in the evaluation of performance and dependability behavior of systems is the assumption that sojourn (holding) time in any state is exponentially distributed. Exponential distribution has many useful properties which lead to analytic tractability, but does not always realistically represent the observed distribution functions. One way to deal with non-exponential distributions is the phase approximation, that is, modeling a distribution by a set of states and transitions between those states such that the holding time in each state is exponentially distributed. The simplest examples of phase approximation are the hyperexponential distribution with a coefficient of variation larger than one, and hypoexponential distribution with a coefficient of variation less than one. Although the method of phase approximation enable

us to use the CTMC model, its major drawback is that it usually results in a large state space.

If transition rates in CTMC are allowed to be time dependent, where time is measured from the beginning of system operation, the model becomes non-homogeneous CTMC. Such models are used in software reliability modeling and in hardware reliability models of non-repairable systems.

Due to the assumptions that holding times in the state are exponentially distributed and that past behavior of the process is completely summarized by the current state of the process, every state transition in a homogeneous CTMC acts as a regeneration point for the process. The first assumption can be generalized by allowing the holding time to have any distribution, thus resulting in the semi Markov process (SMP). The seconds assumption can also be generalized by allowing not all state transitions to be regeneration points, thereby resulting in the Markov regenerative process (MRGP). For the mathematical definitions of these stochastic processes the reader is referred to [9].

7 Generation Techniques for State Space Models

We have already pointed out the importance of high level specification and automated generation of large Markov chains. The approach of separation of higher level model description and lower level computational model has many advantages. Besides reducing the size of the description, such models provide visual and conceptual clarity, and are closer to a designer's intuition. They allow the designer to focus more on the system being modeled rather than on error-prone and tedious creation of lower level models manually.

In this section we will briefly review the stochastic Petri net models and their extensions, whose popularity is partially due to the number of software tools available for their specification and analysis; these include SHARPE [14] and SPNP [6].

As originally introduced by C.A.Petri in 1962, Petri nets did not have a time element. A stochastic Petri nets (SPN) is a Petri net with timed transitions where the firing time distributions are assumed to be exponential. A generalized stochastic Petri nets (GSPN), first proposed in [1], is a Petri net where both immediate and timed transitions are allowed. The underlying stochastic process of the SPNs and GSPNs is a Markov chain.

In the last decade many extensions to the basic Petri net model have been proposed. Some of these extensions have enhanced the flexibility of use and allowed even more concise description of performance and dependability models. Some other extensions have enhanced the modeling power by allowing for a reward rate functions or non-exponential distributions. Specifically, besides several structural extensions, Stochastic Reward Nets (SRN) allow a reward rate to be associated with each reachable marking. SRNs have been shown to be isomorphic to Markov reward processes. The Extended Stochastic Petri net (ESPN), in which general firing time distributions are allowed, under suitable conditions has as the underlying stochastic process a semi Markov process. Deterministic

Stochastic Petri nets (DSPN) allow the definition of immediate, exponential and deterministic transitions. The stochastic process underlying a DSPN is a Markov regenerative process. The Markov regenerative Stochastic Petri nets (MRSPN) generalize DSPNs and still have MRGP as an underlying stochastic process. The Concurrent Generalized Petri nets (CGPN) allow simultaneous enabling of any number of immediate, exponentially distributed and generally distributed timed transitions, provided that the latter are all enabled at the same instant. Stochastic process underlying a CGPN is also MRGP.

To summarize, the use of SPNs and their extensions for a high level specification of stochastic models has received a lot attention in current research. An excellent survey could be found in [10], [4].

8 Conclusion

Analytical modeling is a cost effective method for examining the behavior of current day complex computer systems. To be useful, the model should be realistic and reflect important system characteristics such as failure behavior, reconfiguration and repair, fault tolerance, graceful degradation, contention for resources, concurrency and synchronization, and deadlines imposed on tasks. The recent advances in the development of different model types, exact and approximate solution techniques, and tools that help in automatic model generation and solution from a high level description enable large and realistic models to be developed and studied effectively.

In this chapter we have discussed the use of analytical models for performance, dependability and performability analysis of computer systems. The choice of an appropriate model type and measures of interest clearly depends on the system requirements. For some systems performance and dependability can be addressed separately, while for others such an analysis can lead to incomplete or even misleading results thus making it essential to combine them in a single framework. A modeler who is familiar with many different types of models, can easily choose the models and measures that best suit a particular system.

At the end we would like to emphasize that analytical modeling is not an exclusive technique for computer system evaluation. Model composition approaches, such as Markov reward models, are particularly suitable for combining different types of models, as well as analytical modeling with measurement or discrete event simulation.

References

1. Ajmone Marsan, M., Balbo,G., Conte, G.: A Class of Generalized Stochastic Petri Nets for the Performance Analysis of Multiprocessor Systems. ACM Trans. on Computer Systems. **2** (1984) 93–122
2. Beaudry, M.D.: Performance – related Reliability Measures for Computing Systems. IEEE Trans. on Computers. **27** (1978) 540–547

3. Bobbio, A., Trivedi, K.: An Aggregation Technique for the Transient Analysis of Stiff Markov Chains. IEEE Trans. on Computers. **35** (1986) 803–814
4. Bobbio, A., Puliafito, A., Telek, M., Trivedi, K.: Recent Developments in Stochastic Petri Nets. Journal of Circuits, Systems, and Computers. **8** (1998) 119–158
5. Bolch, G., Greiner, S., de Meer, H.. Trivedi, K.: Queueing Networks and Markov Chains, John Wiley & Sons, New York (1998)
6. Ciardo, G., Blakemore, A., Chimento, P.F., Muppala, J.K., Trivedi, K.S.: Automated Generation and Analysis of Markov Reward Models using Stochastic Reward Nets. In Mayer, C., Plemmons, R.J. (eds.): Linear Algebra, Markov Chains and Queueing Models. IMA Volumes in Mathematics and Its Applications. Vol.48. Springer-Verlag (1993) 145–191
7. Gross, D., Harris, C.M.: Fundamentals of Queueing Theory. John Wiley & Sons, New York (1985)
8. Howard, R.A.: Dynamic Probabilistic Systems, Vol.II: Semi–Markov and Decision Processes. John Wiley and Sons, New York (1971)
9. Kulkarni, V.G.: Modeling and Analysis of Stochastic Systems. Chapman Hall (1995)
10. Lindemann, C.: Stochastic Modeling using DSPNexpress. Oldenburg, Munich (1994)
11. Meyer, J.F.: On Evaluating the Performability of Degradable Computing Systems. IEEE Trans. on Computers. **29** (1980) 720–731
12. Trivedi, K.S.: Probability and Statistics with Reliability, Queuing and Computer Science Applications. Prentice–Hall, Englewood Cliffs, New Jersey (1982)
13. Trivedi, K.S., Sathaye, A.S., Ibe, O.C., Howe, R.C., Aggarwal, A.: Availability and Performance – Based Sizing of Multiprocessor Systems, Communications in Reliability, Maintainability and Serviceability. (1996)
14. Sahner, R.A., Trivedi, K.S., Puliafito, A.: Performance and Reliability Analysis of Computer Systems. Kluwer Academic Publishers, Norwell (1996)